Command Handler (encrypted)

Player.communicator — send message - relevant msg at — add message relue — communicator — add message relue — send message

Issue command

Issue Events

Communication Manager
Decode & Dispatch()
*event-Listener
Encode & send message

add event listener
2 Update

Communication RuleBooks
communication dispatch rule

add event listener

Event Manager/Event Handler

cards
* Event listener

Update
add event listener

Update
2

gamePhase
* Event listener

game cards encapsulate card
card - all state info outside of game
Game card differences
• Position on game area
• Master & owner
• (stage) orientation   sideways face-up/down
• abilities registered w/ game
• GUID (game-unique-ID)

Removecard [PlayerID] [CardID] [Zone]

ADDCARD [PlayerID] [CardID] [Zone Name]

Event Move [src playerID] [src zone name] [dst playerID] [dst zone Name] [card ID]

Event DRAWCARD [PlayerID] (move from Deck to hand)

* Anytime a player has to interact w/ a card, it's by the
collectionID. GUID remains Internal)
 - Tell a zone to act, not a card
 - ask for a collectionID to play a card or retrieve from
   zone
 - If possible Interact w/ zone, not card to maintain Info control

INFO SEARCH // requires a Builder
      ZONE [ownerID] [zone Name] - all Relevant card info
      CARD [card ID] - static Basic Info of card

Info From INFO [requestID] [Info keyword] [keyword args...]

INFORESPONSE [requestID] [response args...]
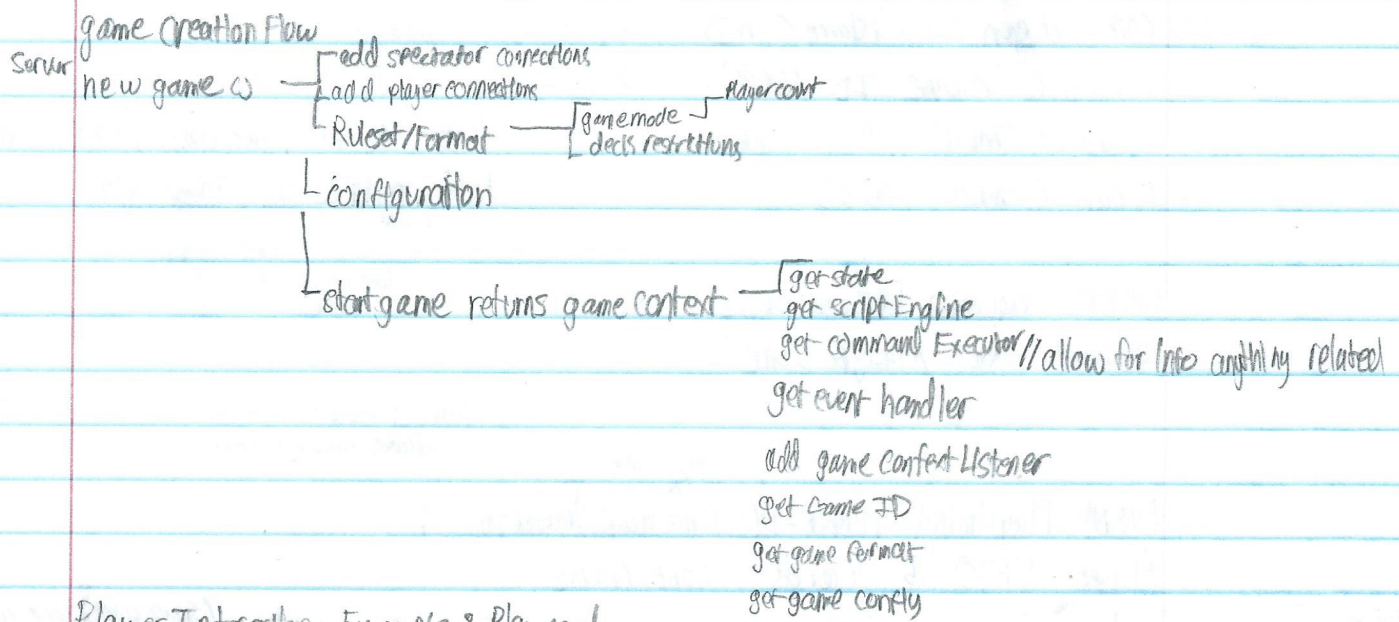   (zone response) [#cards]   [json of card info
                                            Unknown * (if hidden)

ID, power, traits, soul, level,
Trigger, cardName*
Card Type, orientation
* only if SenderType is card

for each card:

game creation flow

Server | new game ()
- add spectator connections
- add player connections
- Ruleset/Format
  - game mode
    - player count
  - decis restrictions
- configuration

- start game returns game context
  - get state
  - get script Engine
  - get command Executor // allow for Info anything related
  - get event handler
  - add game context Listener
  - get Game ID
  - get game format
  - get game config

Player Interaction Example: Play card
Assume: main phase play timing
I. Player selects

Player communications process

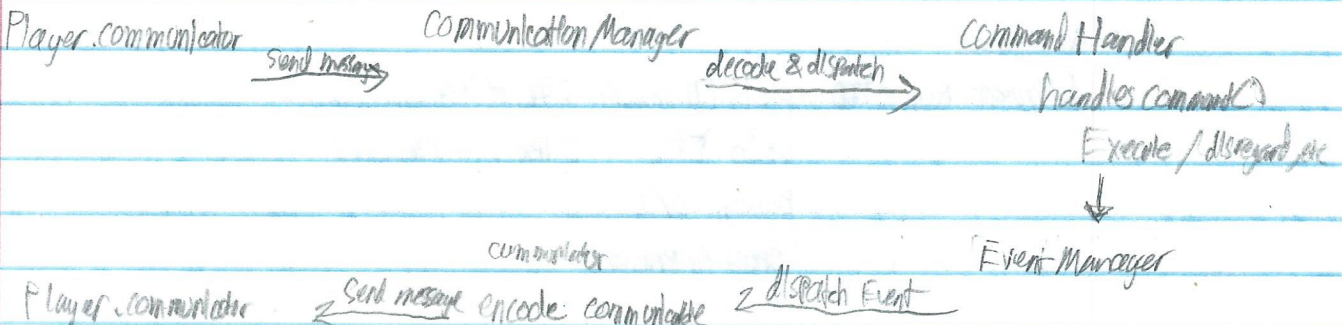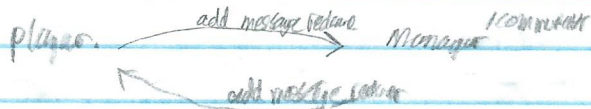1st Idea | send Message (message)   Recieve Message ()

name parameters   Description   Example
ADD   UUID ZONE_Name

player. —add message redone→ Manager communicat
        ←add message redone—

Concurrency
Event: Reshuffle, reason for issue
       dispatch

Player.communicator   Communication Manager   Command Handler
            —send message→   —decode & dispatch→   handles command()
                                                                     Execute / disregard, etc

                                                                        ↓
                                communicator   Event Manager
Player.communicator   —send message encode: communicate→   —dispatch Event—

Event Play Timing PlayID choose 1 EDis3
INFO

types command sender

c = command gen .new instance (this)   // passed to script engine or sender.card

c.call ("move ID HAND") → call

c.call ("MOVE", "ID HAND")   Find Keyword, gets class, default script   - decoder

c.call ("Move" SomeID)   Factory.create (class, keyword; arglist
                                        | try
                                        | class.new instance (...)
                                        | catch (Illegal args)
                                        | return null

Response RequestID [response args]

Command Attack ATK-Type zone   Executor.execute
                                                creates command (context)

                      only in Event
EVENT PlayTiming PlayerID ([Timestamp]) Request-ID   :

PT-Type UPTO 5 Cards zone_Name   (Search criteria, zone, manual Enumeration
        |‾ Decision Decision-Type [Decision count] [option-Type] |‾ cards   [list of card IDs]
        |_ Phase [Phase-Name] |_ special   [list of options...]

T-Type: all playtimings: Prep Phase response:
phase, Request   Decision pick # cards [list of cards]   Response RequestID Mulligan 0 or null
                                  | special [list of options]   card ID...
Decision-Type:
pick, upto, boolean   upto # cards [list...
option-Type: special   list...
cards, special
                        Boolean   ← - - -

                       Phase   [Phase Name]

                       all responses:
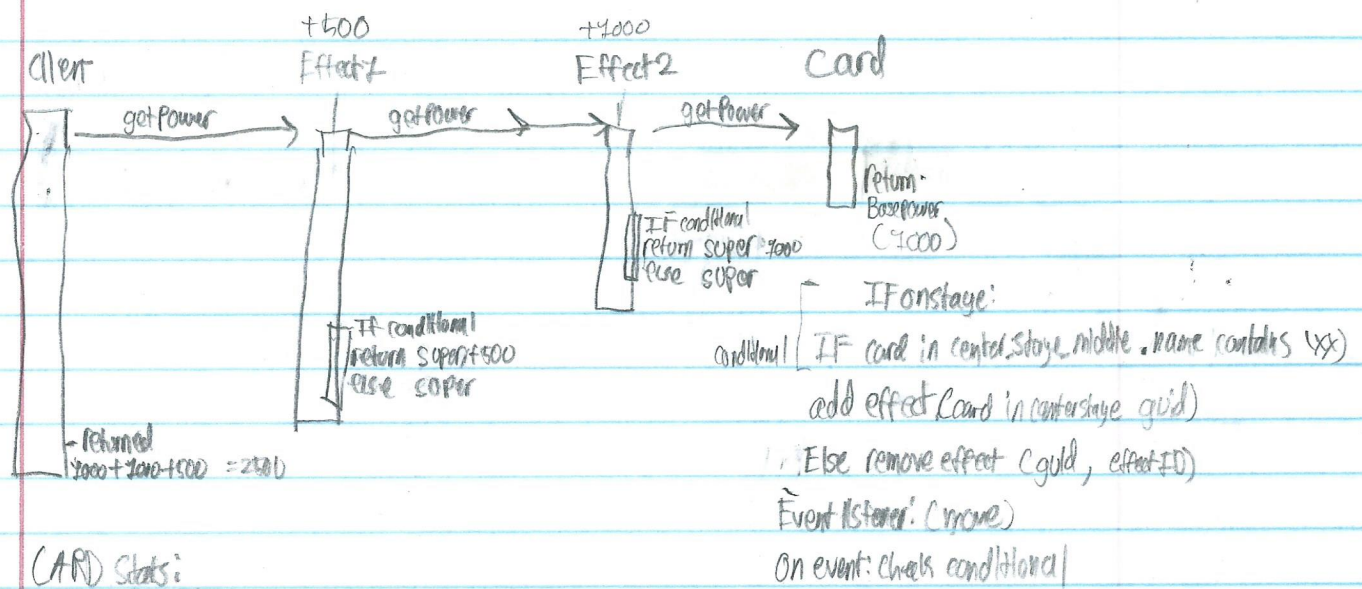   Request RequestID picks [#'s selected] [# of items...
                                           upto [#]   [list of items ...]
                                           Boolean 0/1
                                           [special or phases]

+500   +1000

Client   Effect1   Effect2   Card

getPower →   getPower →   getPower →

Card:
return:
BasePower
(1000)

Effect2:
IF conditional
return super +1000
else super

Effect1:
IF conditional
return super +500
else super

Client:
← returned
1000 + 1000 + 500 = 2500

conditional [IFonstage:
IF card in center.stage.middle. name contains xx)
add effect (card in centerstage guid)
, Else remove effect (guid, effectID)

Event listener: (move)
On event: checks conditional

CARD stats:

← Every stat can be modified with: Apply Effect ([Name of stat], modification type) ([amount] or List Items affecting